[40101/07301 - 2000.034]

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventors	:	Mark A. Stevens
Serial No.	:	09/766,335
Filed	:	January 19, 2001
For	:	Conversion System for Translating Structured Documents into Multiple Target Formats
Group Art Unit	:	2178
Examiner	:	Cong-lac T Huynh

Mail Stop: Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

STATEMENT OF PRIOR SUBMISSION OF IDS

Applicants bring to the Examiner's attention that an Information Disclosure Statement submitted on January 19, 2001 in the above-identified application was not considered prior to the issuance of the Office Action mailed on April 26, 2004 by the Examiner. A copy of the IDS and two (2) references is enclosed for your records.

Respectfully submitted,

Dated: August 11, 2005

By:________________________
Michael J. Marcin, Reg. 48,198

Fay Kaplun & Marcin, LLP
150 Broadway, Suite 702
New York, NY 10038
Tel: (212) 619-6000
Fax: (212) 619-0276

1.97(b) or 1.97(c))

In Re Application Of: Stev			Docket No. 2000.034/1109.007
Serial No.	Filing Date 1/19/00	Examiner	Group Art Unit

Title: Conversion System For Translating Structured Documents Into Multiple Target Formats

Address to:
Assistant Commissioner for Patents
Washington, D.C. 20231

1. ☒ **37 CFR 1.97(b)**

The Information Disclosure Statement submitted herewith is being filed within three months of the filing of a national application; within three months of the date of entry of the national stage as set forth in 37 CFR 1.491 in an international application; or before the mailing date of a first Office Action on the merits, whichever event occurs last.

2. □ **37 CFR 1.97(c)**

The Information Disclosure Statement submitted herewith is being filed after three months of the filing of a national application, or the date of entry of the national stage as set forth in 37 CFR 1.491 in an international application; or after the mailing date of a first Office Action on the merits, whichever occurred last but before the mailing date of either:

1. a Final Action under 37 CFR 1.113, or
2. a Notice of Allowance under 37 CFR 1.311,

whichever occurs first.

Also submitted herewith is:

□ a certification as specified in 37 CFR 1.97(e);

OR

□ the fee set forth in 37 CFR 1.17(p) for submission of an Information Disclosure Statement under 37 CFR 1.97(c).

1.97(b) or 1.97(c))

In Re Application Of: Steven

Docket No. 2000.034/1109.007

Serial No.	Filing Date	Examiner	Group Art Unit
	1/19/00		

Title: Conversion System For Translating Structured Documents Into Multiple Target Formats

Payment of Fee

(Only complete if Applicant elects to pay the fee set forth in 37 CFR 1.17(p))

□ A check in the amount of   is attached.

☒ The Assistant Commissioner is hereby authorized to charge and credit Deposit Account No. 50-0734 as described below. A duplicate copy of this sheet is enclosed.

- □ Charge the amount of
- ☒ Credit any overpayment.
- ☒ Charge any additional fee required.

Certificate of Transmission by Facsimile*

I certify that this document and authorization to charge deposit account is being facsimile transmitted to the United States Patent and Trademark Office (Fax. No.   ) on
(Date)

Signature

Typed or Printed Name of Person Signing Certificate

*This certificate may only be used if paying by deposit account.

Certificate of Mailing by First Class Mail

I certify that this document and fee is being deposited on   with the U.S. Postal Service as first class mail under 37 C.F.R. 1.8 and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

Signature of Person Mailing Correspondence

Typed or Printed Name of Person Mailing Correspondence

Signature

Dated: 1/19/01

Richard L. Sampson
Attorney for Applicant
Reg. No. 37,231

Best Available Copy

CC:

Sheet 1 of 1

INFORMATION DISCLOSURE CITATION

Docket No. 2000.034/1109.007

Serial No.

Applicant: Stevens

Filing Date: 1/19/01

Group:

U.S. PATENT DOCUMENTS

Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date If Appropriate

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	Date	Country	Class	Subclass	Translation	
							Yes	No

Other Documents (including Author, Title, Date, pertinent public. etc.)

	1.	yacc.txt, 5 pages
	2.	lex.txt, 17 pages

Examiner

Date Considered

```
User Commands
                                                   yacc(1)

NAME
     yacc - yet another compiler-compiler

SYNOPSIS
     /usr/ccs/bin/yacc  [ -dltVv ]  [ -b file_prefix ]  [ -Q
      [y  | n ]   ]   [ -P parser ]  [ -p sym_prefix ] file

DESCRIPTION
     The yacc  command converts a context-free grammar into a set
     of  tables  for  a simple automaton that executes an LALR(1)
     parsing algorithm. The grammar may be  ambiguous;  specified
     precedence rules are used to break ambiguities.

     The output file, y.tab.c, must be compiled by the C compiler
     to produce a function yyparse(). This program must be loaded
     with the lexical  analyzer  program,  yylex(),  as  well  as
     main()  and yyerror(), an error handling routine. These rou-
     tines must be supplied by the user; the  lex(1)  command  is
     useful for creating lexical analyzers usable by yacc .

OPTIONS
     The following options are supported:

     -b _□f_□i_□l_□e__□p_□r_□e_□f_□i_□x
               Use _□f_□i_□l_□e__□p_□r_□e_□f_□i_□x instead of y as the prefix fo
r all

               output  files.   The code file y.tab.c, the header
               file y.tab.h (created when -d is  specified),  and
               the  description file y.output (created when -v is
               specified), will be changed to  _□f_□i_□l_□e__□p_□r_□e_□f_□i_□x.t
ab.c,
               _□f_□i_□l_□e__□p_□r_□e_□f_□i_□x.tab.h, and _□f_□i_□l_□e__□p_□r_□e
_□f_□i_□x.output, respec-
               tively.

     -d        Generates the file y.tab.h with the #define state-
               ments   that  associate  the  yacc   user-assigned
               "token  codes"  with  the   user-declared   "token
               names." This association allows source files other
               than y.tab.c to access the token codes.

     -l        Specifies that the code produced in  y.tab.c  will
               not  contain  any  #line  constructs.  This option
               should only be used  after  the  grammar  and  the
               associated actions are fully debugged.

     -P _□p_□a_□r_□s_□e_□r Allows you to specify the parser  of  your  choice
               instead  of /usr/ccs/bin/yaccpar. For example, you
               can specify:
```

```
          example% yacc -P ~/myparser parser.y

          -p _□s_□y_□m__□p_□r_□e_□f_□i_□x
                        Use _□s_□y_□m__□p_□r_□e_□f_□i_□x instead of yy as the prefix for
all
                        external  names  produced  by  yacc  .  The  names

SunOS 5.7           Last change: 20 Dec 1996                 1



User Commands
                                                             yacc(1)

                        affected include the functions yyparse(),  yylex()
                        and  yyerror(),  and  the variables _□y_□y_□l_□v_□a_□l, _□y_□y_□c
_□h_□a_□r
                        and _□y_□y_□d_□e_□b_□u_□g. (In the remainder  of  this  section,
                        the  six  symbols cited are referenced using their
                        default names only as a  notational  convenience.)
                        Local names may also be affected by the -p option;
                        however, the -p option  does  not  affect  #define
                        symbols generated by yacc .

          -Q[y|n]       The -Qy option puts the version stamping  informa-
                        tion in y.tab.c. This allows you to know what ver-
                        sion of yacc  built the file. The -Qn option  (the
                        default) writes no version information.

          -t            Compiles runtime debugging code by default.   Run-
                        time debugging code is always generated in y.tab.c
                        under conditional compilation control. By default,
                        this  code  is  not  included when y.tab.c is com-
                        piled. Whether or not the -t option is  used,  the
                        runtime  debugging  code  is  under the control of
                        YYDEBUG , a preprocessor symbol. If YYDEBUG has  a
                        non-zero   value,   then  the  debugging  code  is
                        included. If its value is 0, then  the  code  will
                        not  be included. The size and execution time of a
                        program produced  without  the  runtime  debugging
                        code will be smaller and slightly faster.

          -V            Prints on the standard error  output  the  version
                        information for yacc .

          -v            Prepares  the  file  y.output,  which  contains  a
                        description  of the parsing tables and a report on
                        conflicts generated by ambiguities in the grammar.

OPERANDS
     The following operand is required:
```

_Of_Oi_Ol_Oe   A path name of a file containing instructions for which a parser is to be created.

EXAMPLES

Example 1: Using The yacc Command

Access to the yacc library is obtained with library search operands to cc. To use the yacc library main,

```
example% cc y.tab.c -ly
```

Both the lex library and the yacc library contain main. To access the yacc main,

SunOS 5.7   Last change: 20 Dec 1996   2

User Commands   yacc(1)

```
example% cc y.tab.c lex.yy.c -ly -ll
```

This ensures that the yacc library is searched first, so that its main is used.

The historical yacc libraries have contained two simple functions that are normally coded by the application programmer. These library functions are similar to the following code:

```
 #include <locale.h>
int main(void)
{
        extern int yyparse();

        setlocale(LC_ALL, "");

        /* If the following parser is one created by lex, the
           application must be careful to ensure that LC_CTYPE
           and LC_COLLATE are set to the POSIX locale.  */
        (void) yyparse();
        return (0);
}

#include <stdio.h>

int yyerror(const char *msg)
{
```

```
                (void) fprintf(stderr, "%s\n", msg);
                return (0);
        }
```

ENVIRONMENT VARIABLES

See environ(5) for descriptions of the following environment variables that affect the execution of yacc : LC_CTYPE, LC_MESSAGES, and NLSPATH.

yacc can handle characters from EUC primary and supplementary codesets as one-token symbols. EUC codes may only be single character quoted terminal symbols. yacc expects yylex() to return a wide character (wchar_t) value for these one-token symbols.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

FILES

y.output state transitions of the generated parser

y.tab.c source code of the generated parser

y.tab.h header file for the generated parser

yacc.acts temporary file

yacc.debug
temporary file

yacc.tmp temporary file

yaccpar parser prototype for C programs

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE

```
|______________________________________□|_____________________________□|
| Availability                          | SUNWbtool                   |
|______________________________________□|_____________________________□|
```

SEE ALSO

cc(1B), lex(1), attributes(5), environron(5)

```
     _□P_□r_□o_□g_□r_□a_□m_□m_□i_□n_□g _□U_□t_□i_□l_□i_□t_□i_□e_□s _□G_□u_□i_□d_
□e
```

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the y.output file. Similarly, if some rules are not reachable from the start symbol, this instance is also reported.

NOTES

Because file names are fixed, at most one yacc process can be active in a given directory at a given time.

User Commands lex(1)

NAME

lex - generate programs for lexical tasks

SYNOPSIS

```
lex  [ -cntv ]  [-e  | -w ]  [ -V -Q
 [y  | n ]  ]  [ file ... ]
```

DESCRIPTION

The lex utility generates C programs to be used in lexical processing of character input, and that can be used as an interface to yacc . The C programs are generated from lex source code and conform to the ISO C standard. Usually, the lex utility writes the program it generates to the file lex.yy.c; the state of this file is unspecified if lex exits with a non-zero exit status. See EXTENDED DESCRIPTION for a complete description of the lex input language.

OPTIONS

The following options are supported:

-c Indicate C-language action (default option).

-e Generate a program that can handle EUC characters (cannot be used with the -w option). yytext[] is of type unsigned char[].

-n Suppress the summary of statistics usually written with the -v option. If no table sizes are specified in the lex source code and the -v option is not specified, then -n is implied.

-t Write the resulting program to standard output instead of lex.yy.c.

-v Write a summary of lex statistics to the standard error. (See the discussion of lex table sizes under the heading Definitions in lex.) If table sizes are specified in the lex source code, and if the -n option is not specified, the -v option may be enabled.

-w Generate a program that can handle EUC characters (cannot be used with the -e option). Unlike the -e option, yytext[] is of type wchar_t[].

-V Print out version information on standard error.

-Q[y|n] Print out version information to output file lex.yy.c by using -Qy. The -Qn option does not print out version information and is the default.

OPERANDS

The following operand is supported:

file — A pathname of an input file. If more than one such file is specified, all files will be concatenated to produce a single lex program. If no file operands are specified, or if a file operand is -, the standard input will be used.

OUTPUT

Stdout

If the -t option is specified, the text file of C source code output of lex will be written to standard output.

Stderr

If the -t option is specified informational, error and warning messages concerning the contents of lex source code input will be written to the standard error.

If the -t option is not specified:

1. Informational error and warning messages concerning the contents of lex source code input will be written to either the standard output or standard error.

2. If the -v option is specified and the -n option is not specified, lex statistics will also be written to standard error. These statistics may also be generated if table sizes are specified with a % operator in the Definitions in lex section (see EXTENDED DESCRIPTION), as long as the -n option is not specified.

Output Files

A text file containing C source code will be written to lex.yy.c, or to the standard output if the -t option is present.

EXTENDED DESCRIPTION

Each input file contains lex source code, which is a table of regular expressions with corresponding actions in the form of C program fragments.

When lex.yy.c is compiled and linked with the lex library (using the -l l operand with c89 or cc), the resulting program reads character input from the standard input and partitions it into strings that match the given expressions.

When an expression is matched, these actions will occur:

+□o The input string that was matched is left in _□y_□y_□t_□e_□x_□t as a null-terminated string; _□y_□y_□t_□e_□x_□t is either an external character array or a pointer to a character string. As

explained in Definitions in lex, the type can be explicitly selected using the %array or %pointer declarations, but the default is %array.

+□o The external int _□y_□y_□l_□e_□n_□g is set to the length of the matching string.

+□o The expression's corresponding program fragment, or action, is executed.

During pattern matching, lex searches the set of patterns for the single longest possible match. Among rules that match the same number of characters, the rule given first will be chosen.

The general format of lex source is:

_□D_□e_□f_□i_□n_□i_□t_□i_□o_□n_□s %%
_□R_□u_□l_□e_□s %%
_□U_□s_□e_□r _□S_□u_□b_□r_□o_□u_□t_□i_□n_□e_□s

The first %% is required to mark the beginning of the rules (regular expressions and actions); the second %% is required only if user subroutines follow.

Any line in the Definitions in lex section beginning with a blank character will be assumed to be a C program fragment and will be copied to the external definition area of the lex.yy.c file. Similarly, anything in the Definitions in lex section included between delimiter lines containing only %{ and %} will also be copied unchanged to the external definition area of the lex.yy.c file.

Any such input (beginning with a blank character or within %{ and %} delimiter lines) appearing at the beginning of the _□R_□u_□l_□e_□s section before any rules are specified will be written to lex.yy.c after the declarations of variables for the yylex function and before the first line of code in yylex. Thus, user variables local to yylex can be declared here, as well as application code to execute upon entry to yylex.

The action taken by lex when encountering any input beginning with a blank character or within %{ and %} delimiter lines appearing in the _□R_□u_□l_□e_□s section but coming after one or more rules is undefined. The presence of such input may result in an erroneous definition of the yylex function.

Definitions in lex

Definitions in lex appear before the first %% delimiter. Any line in this section not contained between %{ and %} lines and not beginning with a blank character is assumed to define a lex substitution string. The format of these lines

SunOS 5.7 Last change: 22 Aug 1997 3

User Commands lex(1)

is:

_□n_□a_□m_□e _□s_□u_□b_□s_□t_□i_□t_□u_□t_□e

If a _□n_□a_□m_□e does not meet the requirements for identifiers in the ISO C standard, the result is undefined. The string _□s_□u_□b_□-_□s_□t_□i_□t_□u_□t_□e will replace the string { _□n_□a_□m_□e } when it is used in a rule. The _□n_□a_□m_□e string is recognized in this context only when the braces are provided and when it does not appear within a bracket expression or within double-quotes.

In the Definitions in lex section, any line beginning with a % (percent sign) character and followed by an alphanumeric word beginning with either s or S defines a set of start conditions. Any line beginning with a % followed by a word beginning with either x or X defines a set of exclusive start conditions. When the generated scanner is in a %s state, patterns with no state specified will be also active; in a %x state, such patterns will not be active. The rest of the line, after the first word, is considered to be one or more blank-character-separated names of start conditions. Start condition names are constructed in the same way as definition names. Start conditions can be used to restrict the matching of regular expressions to one or more states as described in Regular expressions in lex.

Implementations accept either of the following two mutually exclusive declarations in the Definitions in lex section:

%array   Declare the type of _□y_□y_□t_□e_□x_□t to be a null-terminated character array.

%pointer   Declare the type of _□y_□y_□t_□e_□x_□t to be a pointer to a null-terminated character string.

Note: When using the %pointer option, you may not also use the yyless function to alter _□y_□y_□t_□e_□x_□t.

%array is the default. If %array is specified (or neither %array nor %pointer is specified), then the correct way to make an external reference to _□y_□y_□e_□x_□t is with a declaration of the form:

```
extern char _□y_□y_□t_□e_□x_□t[]
```

If %pointer is specified, then the correct external reference is of the form:

```
extern char *_□y_□y_□t_□e_□x_□t;
```

lex will accept declarations in the Definitions in lex section for setting certain internal table sizes. The

declarations are shown in the following table.

Table Size Declaration in lex

Declaration	Description	Default
%p_□n	Number of positions	2500
%n_□n	Number of states	500
%a _□n	Number of transitions	2000
%e_□n	Number of parse tree nodes	1000
%k_□n	Number of packed character classes	10000
%o_□n	Size of the output array	3000

□

Programs generated by lex need either the -e or -w option to handle input that contains EUC characters from supplementary codesets. If neither of these options is specified, yytext is of the type char[], and the generated program can handle only ASCII characters.

When the -e option is used, yytext is of the type unsigned char[] and yyleng gives the total number of _□b_□y_□t_□e_□s in the matched string. With this option, the macros input(), unput(_□c), and output(_□c) should do a byte-based I/O in the same way as with the regular ASCII lex . Two more variables are available with the -e option, yywtext and yywleng, which behave the same as yytext and yyleng would under the -w option.

When the -w option is used, yytext is of the type wchar_t[] and yyleng gives the total number of _□c_□h_□a_□r_□a_□c_□t_□e_□r_□s in the matched string. If you supply your own input(), unput(_□c), or output(_□c) macros with this option, they must return or accept EUC characters in the form of wide character (wchar_t). This allows a different interface between your program and the lex internals, to expedite some programs.

Rules in lex

The Rules in lex source files are a table in which the left column contains regular expressions and the right column contains actions (C program fragments) to be executed when the expressions are recognized.

_□E_□R_□E _□a_□c_□t_□i_□o_□n_□E_□R_□E _□a_□c_□t_□i_□o_□n ...

The extended regular expression (ERE) portion of a row will be separated from _□a_□c_□t_□i_□o_□n by one or more blank characters. A regular expression containing blank characters is recognized under one of the following conditions:

+□o The entire expression appears within double-quotes.

+□o The blank characters appear within double-quotes or square brackets.

+□o Each blank character is preceded by a backslash char-

```
          acter.

     User Subroutines      in lex
        Anything in the user subroutines section will be  copied  to
        lex.yy.c following yylex.

     Regular Expressions      in lex
        The lex   utility  supports  the  set  of  Extended  Regular
        Expressions  (EREs) described on regex(5) with the following
        additions and exceptions to the syntax:

        ...        Any  string   enclosed   in   double-quotes   will
                   represent  the characters within the double-quotes
                   as  themselves,  except  that  backslash   escapes
                   (which  appear  in the following table) are recog-
                   nized. Any backslash-escape sequence is terminated
                   by   the  closing  quote.  For  example,  "\01""1"
                   represents a single string: the octal value 1 fol-
                   lowed by the character 1.

        <_□s_□t_□a_□t_□e>_□r

        <_□s_□t_□a_□t_□e_□1, _□s_□t_□a_□t_□e_□2, ...>_□r
                   The regular expression _□r will be matched only when
                   the  program  is  in  one  of the start conditions
or more            indicated by _□s_□t_□a_□t_□e, _□s_□t_□a_□t_□e_□1, and so forth; f

                   information see Actions in lex (As an exception to
                   the typographical conventions of the rest of  this
                   document,  in this case <_□s_□t_□a_□t_□e> does not represent
                   a  metavariable,  but  the  literal  angle-bracket
                   characters surrounding a symbol.) The start condi-
                   tion is recognized as such only at  the  beginning
                   of a regular expression.

        _□r/_□x        The regular expression _□r will be matched  only  if
                   it is followed by an occurrence of regular expres-
                   sion _□x. The token returned  in  _□y_□y_□t_□e_□x_□t  will  only
                   match  _□r. If the trailing portion of _□r matches the
                   beginning of _□x, the result is unspecified.  The  _□r
                   expression cannot include further trailing context
                   or the $ (match-end-of-line)  operator;  _□x  cannot
                   include  the ^ (match-beginning-of-line) operator,
                   nor trailing context, nor the $ operator. That is,
                   only one occurrence of trailing context is allowed
                   in a lex  regular expression, and the  ^  operator
                   only  can  be  used  at  the  beginning of such an
                   expression. A  further  restriction  is  that  the
                   trailing-context  operator  /  (slash)  cannot  be
                   grouped within parentheses.
```

```
m      {_□n_□a_□m_□e}    When _□n_□a_□m_□e is one of the substitution symbols  fro
ing the          the _□D_□e_□f_□i_□n_□i_□t_□i_□o_□n_□s section, the string, includ
                 enclosing braces, will be replaced by the  _□s_□u_□b_□s_□t_□i_□-
                 _□t_□u_□t_□e  value.  The _□s_□u_□b_□s_□t_□i_□t_□u_□t_□e value wi
ll be treated
                 in the extended regular expression as if  it  were
                 enclosed  in  parentheses.  No  substitution  will
                 occur if {_□n_□a_□m_□e} occurs within a bracket expression
                 or within double-quotes.

     Within an ERE, a backslash character (\\, \a,  \b,  \f,  \n,
     \r,  \t,  \v)  is considered to begin an escape sequence. In
     addition, the escape sequences in the following  table  will
     be recognized.

     A literal newline character cannot occur within an ERE;  the
     escape  sequence \n can be used to represent a newline char-
     acter. A newline character cannot be  matched  by  a  period
     operator.

     Escape Sequences in lex
     ___________________________________________________________________
___ |
   |                            Escape Sequences
    | in lex
   |
    | Escape Sequence   Description
   |                                                  Meaning
   | \_□d_□i_□g_□i_□t_□s              A  backslash  character  fol-   The characte
r whose  encod-|
    |                    lowed by the longest sequence   ing  is  represented by
the|
    |                    of one, two or  three  octal-   one-, two-  or  three-di
git|
    |                    digit  characters (01234567).   octal  integer.  Multi-b
yte|
    |                    Ifall of the  digits  are  0,   characters  require  mul
ti-|
    |                    (that  is,  representation of   ple,   concatenated  esc
ape|
    |                    the  NUL  character),   the    sequences  of  this  ty
pe,|
    |                    behavior is undefined.         including the leading \
for|
    |
   |                                                    each byte.
   | \x_□d_□i_□g_□i_□t_□s            A  backslash  character  fol-   The characte
```

```
r whose  encod-|
     |
the|                     lowed by the longest sequence   ing  is  represented by
     |
   |                     of hexadecimal-digit  charac-   hexadecimal integer.
     |
   |                     ters   (01234567abcdefABCDEF).
     |
   |                     If all of the digits  are  0,
     |
   |                     (that   is,  representation of
     |
   |                     the    NUL   character),   the
     |
   |                     behavior is undefined.
     |
     | \_Oc
nged.|                     A  backslash  character  fol-   The character c, uncha
     |
   |                     lowed  by  any  character not
     |
   |                     described  in   this   table.
     |
   |                     (\\, \a, \b, \f, \en, \r, \t,
     |
   |                     \v).
     |
     |____________________________________________________________________
____O|
```

The order of precedence given to extended regular expressions for lex is as shown in the following table, from high to low.

Note: The escaped characters entry is not meant to imply that these are operators, but they are included in the table to show their relationships to the true operators. The start condition, trailing context and anchoring notations have been omitted from the table because of the placement restrictions described in this section; they can only appear at the beginning or ending of an ERE.

```
|______________________________________________________________
| in lex                          ERE Precedence                |
                                                                 |
                           | _□c_□o_□l_□l_□a_□t_□i_□o_□n-_□r_□e_□l_□a_□t_□e_□d _□b_□r_□a_□c_□
k_□e_□t  _□s_□y_□m_□b_□o_□l_□s      [= =]   [: :]   [. .]
                           | _□e_□s_□c_□a_□p_□e_□d  _□c_□h_□a_□r_□a_□c_□t_□e_□r_□s      |
     \<_□s_□p_□e_□c_□i_□a_□l  _□c_□h_□a_□r_□a_□c_□t_□e_□r>
                           | _□b_□r_□a_□c_□k_□e_□t  _□e_□x_□p_□r_□e_□s_□s_□i_□o_□n      |
     [ ]                                    |
                           | _□q_□u_□o_□t_□i_□n_□g
                          |                                                  "..."
                           | _□g_□r_□o_□u_□p_□i_□n_□g
                            |                                                 ()
□m_□e}                     | _□d_□e_□f_□i_□n_□i_□t_□i_□o_□n
                                       |                                          {_□n_□a_
                           | _□s_□i_□n_□g_□l_□e-_□c_□h_□a_□r_□a_□c_□t_□e_□r _□R_□E _□d_□u_□p_
□l_□i_□c_□a_□t_□i_□o_□n         * + ?
                           | _□c_□o_□n_□c_□a_□t_□e_□n_□a_□t_□i_□o_□n           |
                                    |
                           | _□i_□n_□t_□e_□r_□v_□a_□l  _□e_□x_□p_□r_□e_□s_□s_□i_□o_□n
      {_□m,_□n}                                |
                           | _□a_□l_□t_□e_□r_□n_□a_□t_□i_□o_□n
                                |                                                 |
□|                         |________________________________________________________
```

The ERE anchoring operators (^ and $) do not appear in the table. With lex regular expressions, these operators are restricted in their use: the ^ operator can only be used at the beginning of an entire regular expression, and the $ operator only at the end. The operators apply to the entire regular expression. Thus, for example, the pattern (^abc)|(def$) is undefined; it can instead be written as two separate rules, one with the regular expression ^abc and one with def$, which share a common action via the special | action (see below). If the pattern were written ^abc|def$, it would match either of abc or def on a line by itself.

Unlike the general ERE rules, embedded anchoring is not allowed by most historical lex implementations. An example of embedded anchoring would be for patterns such as (^)foo($) to match foo when it exists as a complete word. This functionality can be obtained using existing lex features:

```
^foo/[ \n]|
" foo"/[ \n]      /* found foo as a separate word */
```

Note also that $ is a form of trailing context (it is equivalent to /\n and as such cannot be used with regular expressions containing another instance of the operator (see the preceding discussion of trailing context).

The additional regular expressions trailing-context operator / (slash) can be used as an ordinary character if presented within double-quotes, "/"; preceded by a backslash, \/; or within a bracket expression, [/]. The start-condition < and > operators are special only in a start condition at the beginning of a regular expression; elsewhere in the regular expression they are treated as ordinary characters.

The following examples clarify the differences between lex regular expressions and regular expressions appearing elsewhere in this document. For regular expressions of the form _□r/_□x, the string matching _□r is always returned; confusion may arise when the beginning of _□x matches the trailing portion of _□r. For example, given the regular expression a*b/cc and the input aaabcc, _□y_□y_□t_□e_□x_□t would contain the string aaab on this match. But given the regular expression x*/xy and the input xxxy, the token xxx, not xx, is returned by some implementations because xxx matches x*.

In the rule ab*/bc, the b* at the end of _□r will extend _□r's match into the beginning of the trailing context, so the result is unspecified. If this rule were ab/bc, however, the rule matches the text ab when it is followed by the text bc . In this latter case, the matching of _□r cannot extend into the beginning of _□x, so the result is specified.

Actions in lex

The action to be taken when an ERE is matched can be a C program fragment or the special actions described below; the program fragment can contain one or more C statements, and can also include special actions. The empty C statement ; is a valid action; any string in the lex.yy.c input that matches the pattern portion of such a rule is effectively ignored or skipped. However, the absence of an action is not valid, and the action lex takes in such a condition is undefined.

The specification for an action, including C statements and special actions, can extend across several lines if enclosed in braces:

```
ERE <one or more blanks> { program statement

program statement }
```

The default action when a string in the input to a lex.yy.c program is not matched by any expression is to copy the string to the output. Because the default behavior of a program generated by lex is to read the input and copy it to the output, a minimal lex source program that has just %%

generates a C program that simply copies the input to the output unchanged.

SunOS 5.7 Last change: 22 Aug 1997 9

User Commands lex(1)

Four special actions are available:

| ECHO; REJECT; BEGIN

| The action | means that the action for the next rule is the action for this rule. Unlike the other three actions, | cannot be enclosed in braces or be semicolon-terminated; it must be specified alone, with no other actions.

ECHO; Write the contents of the string _□y_□y_□t_□e_□x_□t on the output.

REJECT; Usually only a single expression is matched by a given string in the input. REJECT means "continue to the next expression that matches the current input," and causes whatever rule was the second choice after the current rule to be executed for the same input. Thus, multiple rules can be matched and executed for one input string or overlapping input strings. For example, given the regular expressions xyz and xy and the input xyz, usually only the regular expression xyz would match. The next attempted match would start after z. If the last action in the xyz rule is REJECT , both this rule and the xy rule would be executed. The REJECT action may be implemented in such a fashion that flow of control does not continue after it, as if it were equivalent to a goto to another part of yylex. The use of REJECT may result in somewhat larger and slower scanners.

BEGIN The action:

BEGIN _□n_□e_□w_□s_□t_□a_□t_□e;

switches the state (start condition) to _□n_□e_□w_□s_□t_□a_□t_□e

If the string _□n_□e_□w_□s_□t_□a_□t_□e has not been declared pre viously as a start condition in the Definitions in lex section, the results are unspecified. The initial state is indicated by the digit 0 or the token INITIAL.

The functions or macros described below are accessible to user code included in the lex input. It is unspecified whether they appear in the C code output of lex , or are accessible only through the -l l operand to c89 or cc (the lex library).

int yylex(void)

Performs lexical analysis on the input; this is the primary function generated by the lex

utility. The function returns zero when the end of input is reached; otherwise it returns non-zero values (tokens) determined by the actions that are selected.

int yymore(void)

When called, indicates that when the next input string is recognized, it is to be appended to the current value of _□y_□y_□t_□e_□x_□t rather than replacing it; the value in _□y_□y_□l_□e_□n_□g is adjusted accordingly.

int_□y_□y_□l_□e_□s_□s(_□i_□n_□t n)

Retains _□n initial characters in _□y_□y_□t_□e_□x_□t, NUL-terminated, and treats the remaining characters as if they had not been read; the value in _□y_□y_□l_□e_□n_□g is adjusted accordingly.

int input(void)

Returns the next character from the input, or zero on end-of-file. It obtains input from the stream pointer _□y_□y_□i_□n, although possibly via an intermediate buffer. Thus, once scanning has begun, the effect of altering the value of _□y_□y_□i_□n is undefined. The character read is removed from the input stream of the scanner without any processing by the scanner.

int unput(int _□c)

Returns the character _□c to the input; _□y_□y_□t_□e_□x_□t and _□y_□y_□l_□e_□n_□g are undefined until the next expression is matched. The result of using _□u_□n_□p_□u_□t for more characters than have been input is unspecified.

The following functions appear only in the lex library accessible through the -l l operand; they can therefore be redefined by a portable application:

int yywrap(void)

Called by yylex at end-of-file; the default yywrap always will return 1. If the application requires yylex to continue processing with another source of input, then the application can include a function yywrap, which associates another file with the external variable FILE *_□y_□y_□i_□n and will return a value of zero.

int main(int _□a_□r_□g_□c, char *_□a_□r_□g_□v[])

Calls yylex to perform lexical analysis, then exits. The user code can contain main to perform application-specific operations, calling yylex as applicable.

SunOS 5.7 Last change: 22 Aug 1997 11

User Commands lex(1)

The reason for breaking these functions into two lists is that only those functions in libl.a can be reliably redefined by a portable application.

Except for input, unput and main, all external and static names generated by lex begin with the prefix yy or YY.

USAGE

Portable applications are warned that in the Rules in lex section, an ERE without an action is not acceptable, but need not be detected as erroneous by lex . This may result in compilation or run-time errors.

The purpose of input is to take characters off the input stream and discard them as far as the lexical analysis is concerned. A common use is to discard the body of a comment once the beginning of a comment is recognized.

The lex utility is not fully internationalized in its treatment of regular expressions in the lex source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer interpret the regular expressions given in the lex source according to the environment specified when the lexical analyzer is executed, but this is not possible with the current lex technology. Furthermore, the very nature of the lexical analyzers produced by lex must be closely tied to the lexical requirements of the input language being described, which will frequently be locale-specific anyway. (For example, writing an analyzer that is used for French text will not automatically be useful for processing other languages.)

EXAMPLES

Example 1: Using lex

The following is an example of a lex program that implements a rudimentary scanner for a Pascal-like syntax:

```
 %{
/* need this for the call to atof() below */
#include <math.h>
/* need this for printf(), fopen() and stdin below */
#include <stdio.h>
%}

DIGIT    [0-9]
ID       [a-z][a-z0-9]*
%%

{DIGIT}+                          {
                        printf("An integer: %s (%d)\n", yytext,
                        atoi(yytext));
```

```
                        }

{DIGIT}+"."{DIGIT}*     {
                        printf("A float: %s (%g)\n", yytext,
                        atof(yytext));
                        }

if|then|begin|end|procedure|function     {
                        printf("A keyword: %s\n", yytext);
                        }
```

```
{ID}
                              printf("An identifier: %s\n", yytext);
"+"|"-"|"*"|"/"
                              printf("An operator: %s\n", yytext);
"{"[^}\n]*"}"
                              /* eat up one-line comments */
[ \t\n]+
                              /* eat up white space */
.
                              printf("Unrecognized character: %s\n", yytext);
%%

int main(int argc, char *argv[])
{
                              ++argv, --argc;  /* skip over program name */
                              if (argc > 0)
     yyin = fopen(argv[0], "r");
                              else
                              yyin = stdin;

}                             yylex();
```

ENVIRONMENT VARIABLES

See environ(5) for descriptions of the following environment variables that affect the execution of lex : LC_COLLATE, LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0   Successful completion.

>0   An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWbtool

SEE ALSO

yacc(1), attributes(5), environ(5), regex(5)

NOTES

If routines such as yyback(), yywrap(), and yylock() in .l (ell) files are to be external C functions, the command line to compile a C++ program must define the __EXTERN_C__ macro. For example:

```
example% CC -D__EXTERN_C__ . . . file
```